**IT301 – Parallel Computing**

Assignment – 6

**Name:** Niraj Nandish

**Roll No:** 191IT234

1. Program 1
   1. Observation – When the if() clause is present with the condition of "n<5" then when we enter 5 for the value of "n", the tasks become undeferred and a single thread executes the whole program. But when the if() clause is removed, then the tasks gets executed parallelly.

Text

Description automatically generated

if() clause present

Text

Description automatically generated

if() clause absent

1. Program 2
   1. Text

      Description automatically generatedObservation – We are initializing the array to a size of 500000 with random integer values. In the best-case scenario, the key to be searched is the mid value of the array. Hence, only one comparison will take place. In the worst-case scenario, the key to be searched will not be present in the array. Hence, the entire array has to be traversed. In the average case scenario, the user inputs a number. In all the cases, we see that parallel runtime is lesser than serial runtime.
   2. Code

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#define size 500000

int binary\_search(int *s*, int *e*, int *array*[], int *key*, int *flag*)

{

if (*s* > *e*)

{

return -1;

}

int a, b;

int m = (*s* + *e*) / 2;

if (*array*[m] == *key*)

{

return m;

}

else

{

#pragma *omp* *task* *shared*(*a*)

{

a = binary\_search(*s*, m - 1, *array*, *key*, *flag*);

}

#pragma *omp* *task* *shared*(*b*)

{

b = binary\_search(m + 1, *e*, *array*, *key*, *flag*);

}

#pragma *omp* *taskwait*

if (a < b)

{

return b;

}

else

{

return a;

}

}

return 0;

}

int comp(const void \**a*, const void \**b*)

{

return (\*(int \*)*a* - \*(int \*)*b*);

}

int main()

{

int array[size], x, pos;

double end, start;

for (int i = 0; i < size; i++)

{

array[i] = rand() % 1000;

}

qsort(array, size, sizeof(int), comp);

printf("\n\nBEST CASE: Key searched is in middle position and found immediately.");

x = array[size / 2];

start = omp\_get\_wtime();

pos = binary\_search(0, size, array, x, 1);

end = omp\_get\_wtime();

printf("\nKey found at: %d", pos);

printf("\nValue = %d", array[pos]);

printf("\nSerial runtime : %lf", end - start);

start = omp\_get\_wtime();

pos = binary\_search(0, size, array, x, 0);

end = omp\_get\_wtime();

printf("\n\nKey found at: %d", pos);

printf("\nValue = %d", array[pos]);

printf("\nParallel runtime : %lf", end - start);

printf("\n\n\nWORST CASE: Key searched is not present in array.");

x = -1000;

start = omp\_get\_wtime();

pos = binary\_search(0, size, array, x, 1);

end = omp\_get\_wtime();

printf("\nKey found at: %d", pos);

printf("\nKey = %d", x);

printf("\nSerial runtime : %lf", end - start);

start = omp\_get\_wtime();

pos = binary\_search(0, size, array, x, 0);

end = omp\_get\_wtime();

printf("\n\nKey found at: %d", pos);

printf("\nKey = %d", x);

printf("\nParallel runtime : %lf", end - start);

printf("\n\n\nAVERAGE CASE: Random key inputted by user.");

printf("\nEnter key to be searched: ");

scanf("%d", &x);

start = omp\_get\_wtime();

pos = binary\_search(0, size, array, x, 1);

end = omp\_get\_wtime();

printf("\nKey found at: %d", pos);

printf("\nValue = %d", array[pos]);

printf("\nSerial runtime : %lf", end - start);

start = omp\_get\_wtime();

pos = binary\_search(0, size, array, x, 0);

end = omp\_get\_wtime();

printf("\n\nKey found at: %d", pos);

printf("\nValue = %d", array[pos]);

printf("\nParallel runtime : %lf\n", end - start);

}